A SageMath: A (Very) Short Introduction

Mathematical software is part of the standard toolkit of mathematicians. Throughout this book I will use the program called SageMath [18] (sometimes called *Sage*, but there are various other software systems called 'Sage") to do computations. The goal of this appendix is to provide some basic orientation on the program, including where to get it. SageMath is free; it can be used online or installed on your own machine; it is very powerful, and it takes some time to learn. SageMath incorporates the functionality of many other free mathematics programs, including Maxima, Octave, R, GAP, and GP. SageMath runs either in a browser window or a terminal.

Of course, there are other mathematical software tools. *Desmos*, *Geogebra*, *Maple*, *Mathematica*, and MATLAB are well-known and quite powerful. The people who make *Mathematica* are also behind Wolfram Alpha, which can do lots of mathematics as well. Many students appear to be familiar with *Desmos* and *Geogebra*. These can all do some of the things we need to do, but I have decided to focus on SageMath.

A.1 General Information

SageMath is an ambitious attempt to create powerful mathematical software that is free and open-source. The ultimate ambition, says the Sage home page, is to create "a viable free open source alternative to *Magma*, *Maple*, *Mathematica* and MATLAB." (I'd say that they are very close to that, and in some aspects well beyond.) The development approach emphasizes openness: while William Stein is the leader of the team, contributions have come from across the mathematical community.

SageMath can be used through a web interface, without needing to download and install the program. There are two ways to do that: either SageMath Cell or the more elaborate interface based on projects and notebooks offered by *CoCalc*. Most of my students find SageMath Cell sufficient for everything they need to do.

It is also possible to download and install the program on your own computer. When you do, you can run SageMath in a terminal window or you can run it in your browser. The latter is much like using *CoCalc*, but the program is running on your local machine.

Of the two web interfaces, SageMath Cell is particularly easy to use for small computations. It presents the user with a big blank rectangle where one can type in Sage commands. Below is a button labeled "Evaluate," which does exactly that. The output appears below. The downside is that SageMath just evaluates what is in the

box. It will not remember what you did, so if you define a symbol and then press the "Evaluate" button, you cannot use it again without repeating the definition. There are two very useful features of SageMath Cell that deserve note. First, it works on a tablet or phone. Second, because SageMath incorporates other open-source mathematical software, SageMath Cell will also allow you to use those: right under the big blank rectangle there is a "Language" drop-down that you can use to do this.

The SageMath home page offers a link to create a *CoCalc Instant Worksheet*, but if you plan to use it regularly and want to save your work from one session to the next you should create a (free) account. (There are also paid CoCalc accounts if you want/need more features.) Once you log in to your account, you can create projects, and each project can contain many notebooks. Notebooks allow you to enter lines of Sage code, which are evaluated when you hit Shift-Enter. Definitions and results are remembered within each session.

If you are going to use the program a lot, then you might want to download and install it. It takes quite a bit of space, but having it on your own machine avoids connectivity issues.

The central web site for SageMath is `www.sagemath.com`. You can go there to download the program, find documentation, and so on.

A.2 First Examples

It's time to show you some examples. You can find many more in A Tour of Sage, which is available online. SageMath is built using the Python programming language, so its syntax is similar to Python's.

Everything is done by entering a command and then asking SageMath to evaluate it. In SageMath Cell, you can enter a sequence of commands as well. If you enter

```
3 + 5
```

into SageMath Cell and hit "Evaluate" you will get

```
8
```

in the results box. SageMath Cell typically only prints your last result. If you want to see two results, it's best to ask it explicitly:

```
print(3+5)
print(57.1^100)
```

Then you get two lines:

```
8
4.60904368661396e175
```

(In the last answer, `e175` means $\times 10^{175}$.) You can get some space between the two lines of output by adding `print(" ")` between the two commands.

In the terminal window, SageMath gives you a prompt and you type in your commands and then hit enter. The same example looks like this:

```
sage: 5+3
8
sage: 57.1^100
4.60904368661396e175
```

Most of the examples we give have been done in a terminal window (for one thing, it is easier to cut and paste from the terminal), so you will see prompts and responses.

The CoCalc or Jupyter notebook interface is like the terminal, but it allows for graphical output. You enter commands and hit shift-enter to execute them. If you define an object with a command like `v=vector([1,2,3])` in CoCalc, the terminal, or a notebook running on your own computer, SageMath will remember what `v` is so that you can use it again later.

SageMath requires you to declare that a letter is a variable before using it that way. The only variable it knows by default is x. So, to work with functions of two variables x and y you will need to begin with

```
sage: var('x y')
(x, y)
```

If you use a variable, say d, that hasn't been defined, you get an error message. SageMath error messages are usually long and cryptic, but they key information is usually at the bottom; this one will end with something like "name 'd' is not defined." (A brand-new feature of CoCalc allows you to ask ChatGPT what is wrong with your code.)

Here is one way to define a function (everything in SageMath can be done in lots of different ways).

```
sage: f(x)=x^4
sage: f
x |--> x^4
sage: f(x)
x^4
```

The command `f(x)=x^4` produces no output, but SageMath has been told what you mean by `f`. Notice also that SageMath distinguishes the function `f` (the rule that sends x to x^4) from `f(x)` (the value of f when you plug in x). All the grown-ups do this, but I haven't done it in this book.

You can compute limits.

```
sage: f(x)=(x-2)/(x+1)
sage: limit(f(x),x=1)
-1/2
sage: limit(f(x),x=oo)
1
sage: limit(f(x),x=-1)
Infinity
```

Notice that you can use `oo` (two lower-case os) to mean infinity.

Two ways to compute a derivative:

```
sage: diff(f,x)
x |--> 4*x^3
sage: diff(f(x),x)
4*x^3
```

(On SageMath Cell you need define f each time, remember.) The first one is the derivative *function* f', the second is $f'(x)$. Higher derivatives are easy too. Here's the second derivative:

```
sage: diff(f,x,2)
x |--> 12*x^2
sage: diff(f(x),x,2)
12*x^2
```

The command `diff(f,x,x)` is equivalent to `diff(f,x,2)`. SageMath also uses (in fact, prefers) the object-oriented convention:

```
sage: f.diff(x)
x |--> 4*x^3
sage: f(x).diff(x)
4*x^3
sage: f(x).diff(x,4)
24
```

One advantage of this syntax is that either in the terminal window or in a notebook you can write `f.` and then hit the tab key. SageMath will then open a pop-up window telling you the many things you can do to the function f.

Here's how to integrate:

```
sage: f(x).integrate(x)
1/5*x^5
sage:
sage: f.integrate(x)
x |--> 1/5*x^5
sage: f(x).integrate(x)
1/5*x^5
sage: integrate(f(x),x)
1/5*x^5
sage: integrate(f(x),x,2,3)
211/5
sage: f(x).integrate(x,2,3)
211/5
```

The last one is the definite integral

$$\int_2^3 f(x)\,dx.$$

Let's try a hard one:

```
sage: g(x)=integrate(sqrt(x)*sqrt(1+x),x)
sage: print(g)
x |--> 1/4*((x + 1)^(3/2)/x^(3/2)
   + sqrt(x + 1)/sqrt(x))/((x + 1)^2/x^2
   - 2*(x + 1)/x + 1) - 1/8*log(sqrt(x + 1)/sqrt(x) + 1)
   + 1/8*log(sqrt(x + 1)/sqrt(x) - 1)
```

The first command produces no output, so I used `print` to tell SageMath to show me the result. The big mess is printed out as a single line, but I have added line breaks here.

Notice that that answer uses the function `log`. That means the *natural logarithm*; SageMath goes understand what `ln` means, but it defaults to using `log`, which is what most mathematicians do. Indeed,

```
sage: ln(2)
log(2)
sage: ln(2).n()
0.693147180559945
```

SageMath tries to give you an exact answer when it can, but you can force an approximate answer in decimal form by using the `n` command.

On SageMath Cell or a notebook, you can get better looking output by using `show` instead of `print`. For example, in SageMath Cell if we enter

```
x = var('x')
g(x)=integrate(sqrt(x)*sqrt(1+x), x)
show(g(x))
```

and hit evaluate, we get something like:

$$\frac{\frac{(x+1)^{\frac{3}{2}}}{x^{\frac{3}{2}}}+\frac{\sqrt{x+1}}{\sqrt{x}}}{4\left(\frac{(x+1)^2}{x^2}-\frac{2(x+1)}{x}+1\right)}-\frac{1}{8}\log\left(\frac{\sqrt{x+1}}{\sqrt{x}}+1\right)+\frac{1}{8}\log\left(\frac{\sqrt{x+1}}{\sqrt{x}}-1\right)$$

There is also `latex(f(x))`, which is what I did to get the LaTeX code to typeset the result. (If you use `show` in the terminal, you get the LaTeX code.)

While `show` produces output that is nicer to look at, the output of `print` is easier to cut-and-paste. In general, it's best to ask SageMath to either `print` or `show` the outputs you want to see.

Maybe there's a way to simplify that monster? One way is to try

```
sage: g(x).simplify_full()
1/4*(2*x + 1)*sqrt(x + 1)*sqrt(x) - 1/8*log((sqrt(x + 1)
  + sqrt(x))/sqrt(x)) + 1/8*log((sqrt(x + 1) - sqrt(x))/sqrt(x))
```

(I have added a line break.)

If we ask SageMath to `show` that, we get

$$\frac{1}{4}(2x+1)\sqrt{x+1}\sqrt{x} - \frac{1}{8}\log\left(\frac{\sqrt{x+1}+\sqrt{x}}{\sqrt{x}}\right) + \frac{1}{8}\log\left(\frac{\sqrt{x+1}-\sqrt{x}}{\sqrt{x}}\right)$$

I guess that is a little better. There are various different versions of `simplify`:

```
f.simplify()
f.simplify_full()
f.simplify_log()
f.simplify_trig()
f.simplify_rational()
```

The first one doesn't usually help; the others can help depending on the situation. See the manual for more details.

Sometimes you also want to use `factor()` and `expand()` to get formulas to look right. Occasionally, `simplify` and others will tell you that they need more information (e.g., "is $x > 0$?"). In that case you get a long error message that tells you what to do at the end (most SageMath error messages give their most useful information at the bottom). You can use `assume(x>0)` or similar commands to provide the needed information.

For example, if you wanted SageMath to simplify $\log(\sqrt{x})$ into $\frac{1}{2}\log(x)$, you would need `assume(x>0)`. You might also need to use `simplify_log`. That might help make the formula for $g(x)$ above a little less intimidating.

A.3 Plotting

SageMath can also plot functions. For example

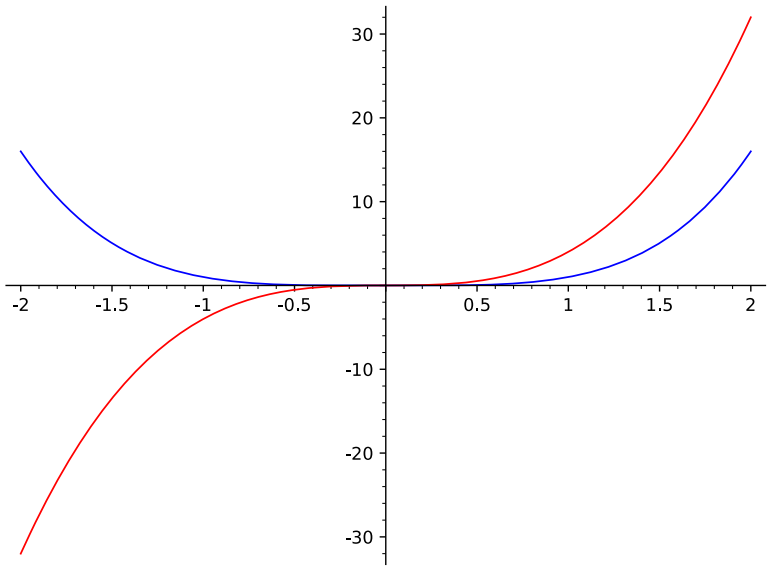
```
sage: plot(f(x),(-2,2))
```

will display a plot of our function. (In a terminal window, SageMath will use your computer's default graphics program to show you the result.) Plotting functions have a zillion options; type `?plot` for documentation and examples. I particularly enjoy playing with the various options related to color.

You can also give your plot a name and then do things to it.

```
sage: P1=plot(f(x),(-2,2),color="blue")
sage: P2=plot(diff(f(x),x),(-2,2),color="red")
sage: P=P1+P2
sage: show(P)
```

Produces this:



How did I get the plot into this LaTeX file? I used

```
sage: P.save('example.eps'}
```

to create an encapsulated postscript (eps) file. The `save` command will use the format indicated by your file name.

There are lots of other plotting commands. To plot a curve given by an equation like $y^2 = x^3 - x$, do this in the cell server:

```
var('x y')
implicit_plot(y^2==x^3-x,(x,-2,2),(y,-2,2))
```

(Note the double equals sign!) Or even in three variables:

```
var('x y z')
implicit_plot3d(y^2*z==x^3-x,(x,-2,2),(y,-2,2),(z,-2,2))
```

This gives a rotatable graph of the surface defined by that equation. The graph displays in a browser window; if you are using the terminal, it will open a new browser window.

I don't know how to save a printable version of a 3D plot, so I can't show you want that looks like. If you figure out a good way to do it, let me know.

A.4 Sums

Since this is a book about sums, it's good to know how to handle them in SageMath. Of course, you can just write them out.

```
sage :1+1/2+1/3+1/4+1/5
137/60
```

That becomes hard to do when you have lots of terms, however. So let's declare a variable *k* and use it to describe the general term of the sum instead:

```
sage: var('k')
k
sage: sum(1/k,k,1,5)
137/60
```

(I won't repeat the definition of *k* again; if you are working with SageMath Cell, it should be done each time.)

Now we can do a larger sum just as easily:

```
sage: sum(1/k,k,1,20)
55835135/1551950
```

SageMath always prefers to give the exact answer, which in this case is a fraction. But we usually want the decimal approximation instead. The easiest way to do this is to make SageMath work with decimals throughout, by replacing 1 with 1.0:

```
sage: sum(1.0/k,k,1,20)
3.597739657143682
```

You can even try an infinite sum:

```
sage: sum(1/k^2,k,1,oo)
1/6*pi^2
```

But that doesn't always work. It certainly fails when the sum does not converge.